ECE385

Fall 2021

Experiment 5

An 8-Bit Multiplier in SystemVerilog

Ge Yuhao, Lou Haina
D231, Oct.27, 2021
TA: Chenhao Wang

1 Introduction

1.1 Summarize the basic functionality of the multiplier circuit

In this experiment, we design a multiplier in SystemVerilog for two 8-bit 2's compliment numbers and then run that multiplier on the DE2 FPGA board. We use a simple add-shift algorithm to multiply two numbers.

2 Pre-lab question

2.1 Rework the multiplication example on page 5.2 of the lab manual, as in compute 00000111 * 11000101 in a table similar to the example

Function	X	A	B	M	Comments for the next step
Clear A, LoadB	0	0000 0000	*0000 0111*	1	Add S to A since M = 1.
ADD	1	1100 0101	*0000 0111*	1	Shift XAB by one bit after ADD complete
SHIFT	1	1110 0010	1*000 0011*	1	Add S to A since M = 1
ADD	1	1010 0111	1*000 0011*	1	Shift XAB by one bit after ADD complete
SHIFT	1	1101 0011	11*00 0001*	1	Add S to A since M = 1
ADD	0	1001 1000	11*00 0001*	1	Shift XAB by one bit after ADD complete
SHIFT	1	1100 1100	011*0 0000*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1110 0110	0011 *0000*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 0011	0001 1*000*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1001	1000 11*00*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1100	1100 011*0*	0	Do not add S to A since M = 0. Shift XAB
SHIFT	1	1111 1110	0110 0011	0	8th shift done. Stop. 16-bit Product in AB

3 Written description and diagrams of multiplier circuit

3.1 Summary of operation

Load To perform a multiplication, we first load the multiplier to Register B by setting the switches (S) to represent the multiplier and pressing the $ClearA\_LoadB$ button. By doing so, it also clears the X and A registers. Then we set the switches (S) to represent the multiplicand S and press the Run button to start computation.

Compute A simple add-shift algorithm is used to multiply B and S. There are several kinds of states in the during the computation. For the ADD state, we first check that if B's lowest bit, namely M is 1, if it is 1, then we simply add or subtract the 9-bits extended version of A and S according to the sign bit X, then store the last 8-bits of the result into A and the first bit in to X, but if M is 0, no adding operation is needed. After the ADD state, we will have a $SHIFT$ state, in which the entire 17 bits of XAB is arithmetically right-shifted by one bit. Then it turns to the ADD state again. The loop will stop only if it reaches the state that there is already 16 $SHIFT$ operations, and then the result is in AB.

Store The circuit should stop once the multiplication is done and the correct result should be displayed by outputting AB on the Hex displays.

3.2 Top Level Block Diagram

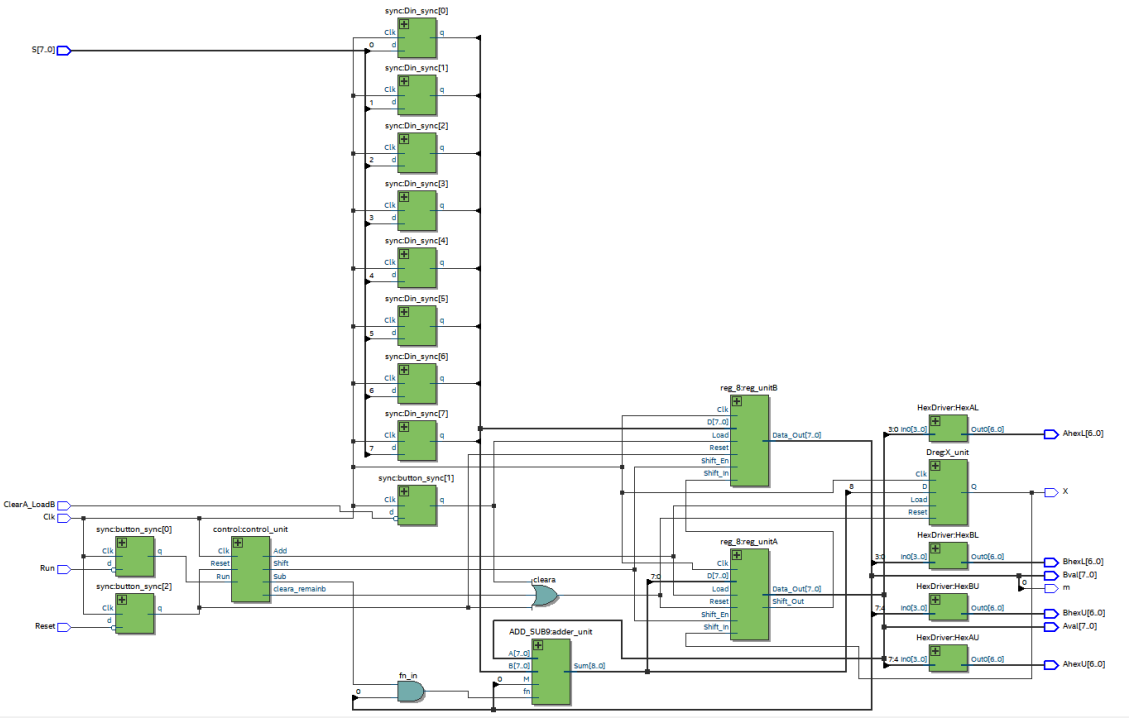

Figure 1: The Top Level Block Diagram from the RTL viewer

3.3 Written Description of all .sv Modules

Module: Processor.sv
Inputs: [7:0]S, Clk, Reset, Run, ClearA_LoadB,
Outputs: [7:0]Aval, [7:0]Bval, [6:0]AhexL, [6:0]AhexU,[6:0]BhexL, [6:0]BhexU, x, m,
Description: This is an 8-bit logic processor top level module.
Purpose: This module serves as the top level moduel which connect all the IO and the circuit.

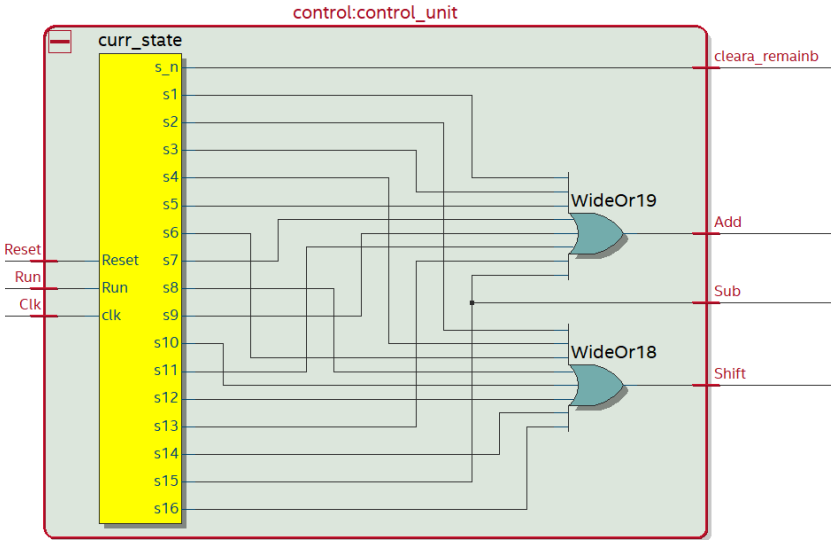

Figure 2: The block diagram for the Processor

Module: ADD_SUB9.sv
Inputs: [7:0]A, [7:0]B, fn, M
Outputs: [8:0] Sum, CO
Description: There is a submodule in this module which is a 4-bit full adder. fn denotes an Addition or Substitution operation, and M denotes whether we condauction an operation between A and B.
Purpose: This is an Addition or Substitution module, we can choose to add A and B or Sub A with B or do nothing according to the input signal.

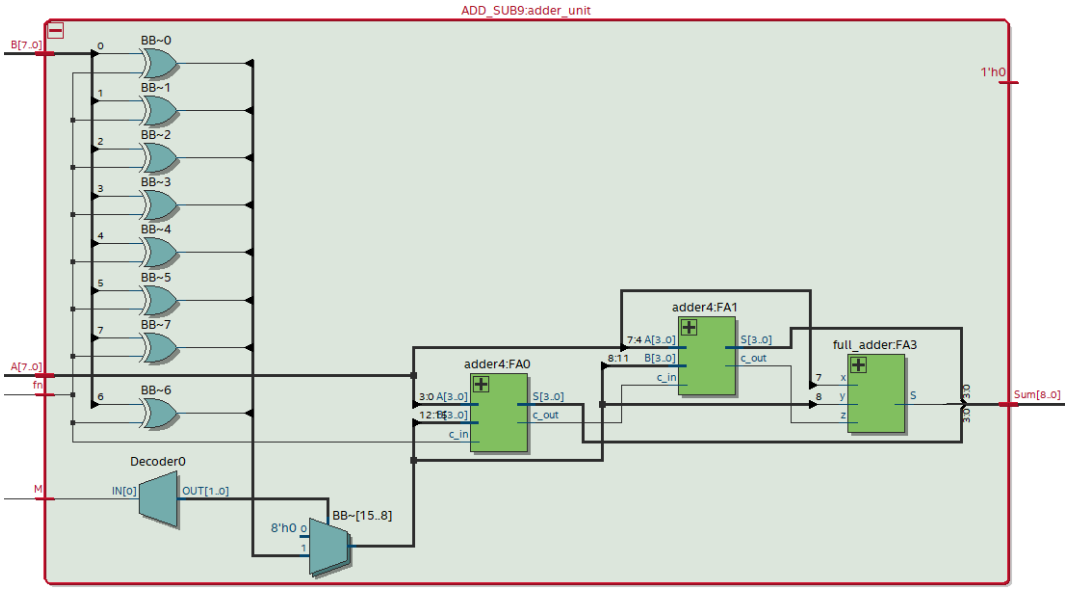

Figure 3: The Block Diagram for ADD$_SUB9$

Module: sync.sv
Inputs: Clk, Reset, d
Outputs: q
Description: An always_ff block is used to bring asynchronous signals into the FPGA.

Purpose: These are synchronizers required for synchronizing the input siganl from button or switches.

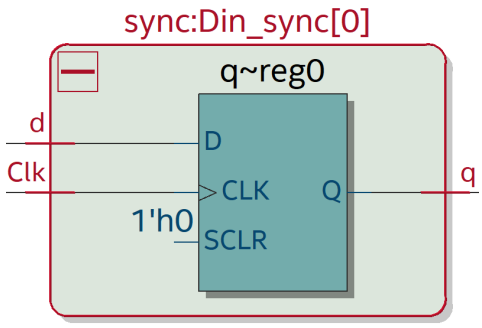

Figure 4: The Top Level Block Diagram from the RTL viewer

Module: reg_8.sv
Inputs: [7:0]D, Clk, Reset, Shift_In, Load, Shift_En
Outputs: [7:0]Data_Out, Shift_Out
Description: This is a positive-edge triggered 8-bit register with asynchronous reset and synchronous load.
Purpose: This module is used to create the registers that store operands A and B in the adder circuit.

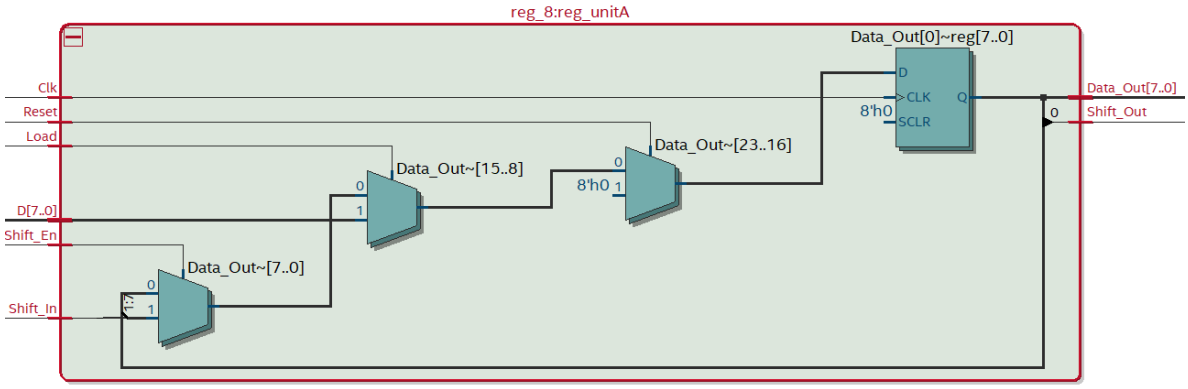

Figure 5: The Top Level Block Diagram from the RTL viewer

Module: Dreg_8.sv
Inputs: Clk, Load, Reset, D
Outputs: Q
Description: This is a D-flip-flop which can store one bit.
Purpose: This module is used to store the sign bit of A.

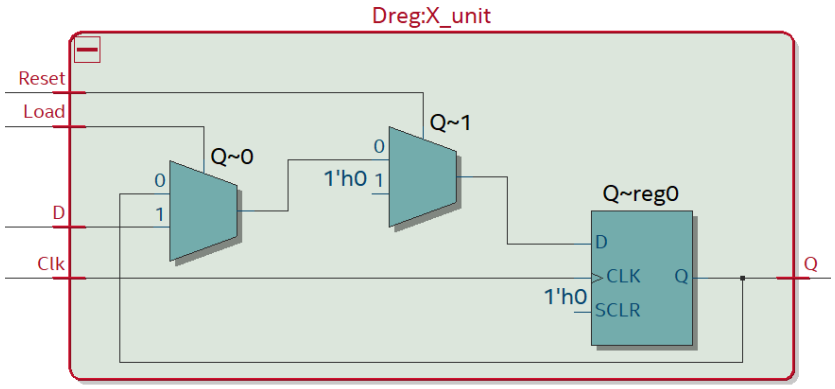

Figure 6: The Top Level Block Diagram from the RTL viewer

Module: HexDriver.sv
Inputs: [3:0]In0
Outputs: [6:0]Out0
Description: This module translate the data to the form that a LED screen can recogize.
Purpose: This module is used to display the value of register A and B onto the LED screen of the FPGA board.

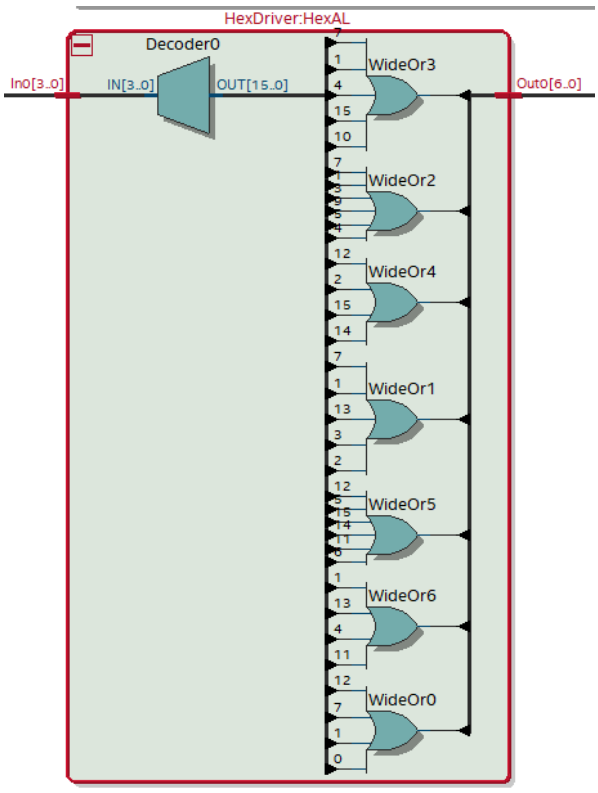

Figure 7: The block diagram for the HexDriver

Module: control.sv
Inputs: Clk, Reset, Run
Outputs: Shift, Add, Sub, cleara_remainb

Description: This module use a Finite State Machine to build a control unit.
Purpose: This module is the control unit which accept input signal from user and give instructions to other unit in the circuit.

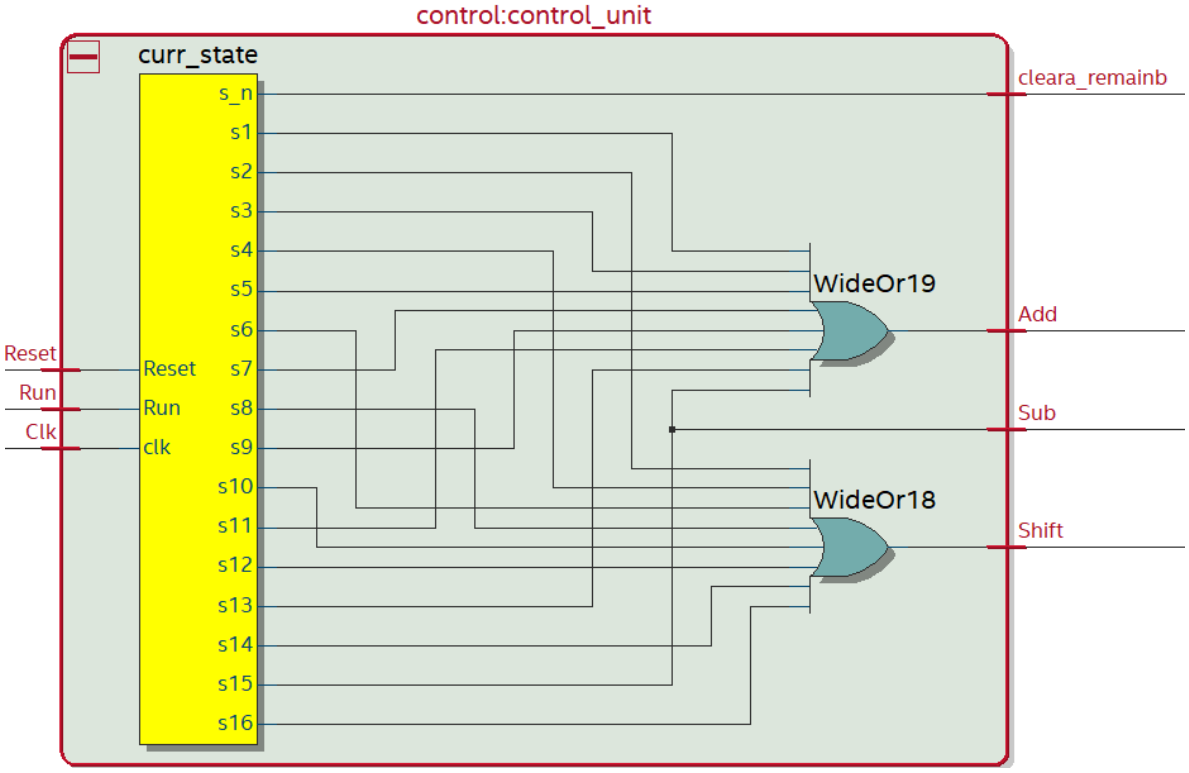

Figure 8: The block diagram for the control unit

3.4 State Diagram for Control Unit

This can be done in a program like Visio, but if the Quartus state diagram generator is used, you must label the states and transitions or you will lose points!

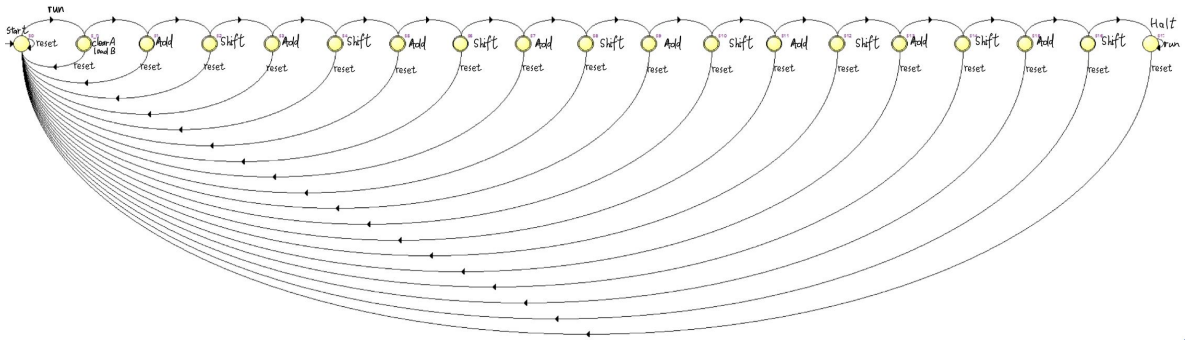

Figure 9: The Finite State Machine

3.5 Annotated pre-lab simulation waveforms (4 simulations: +*+, +*-, -*+, -*-)

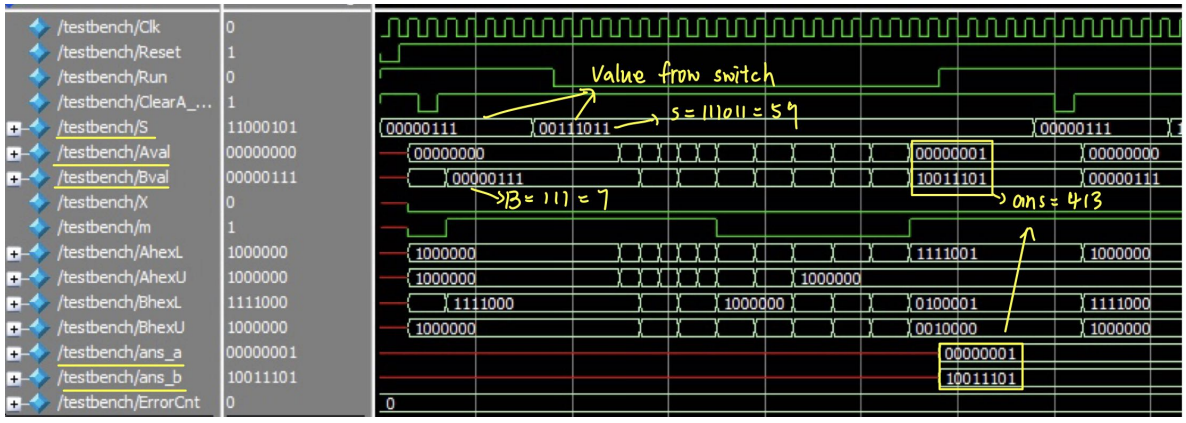

Figure 10: Waveform of 59*7

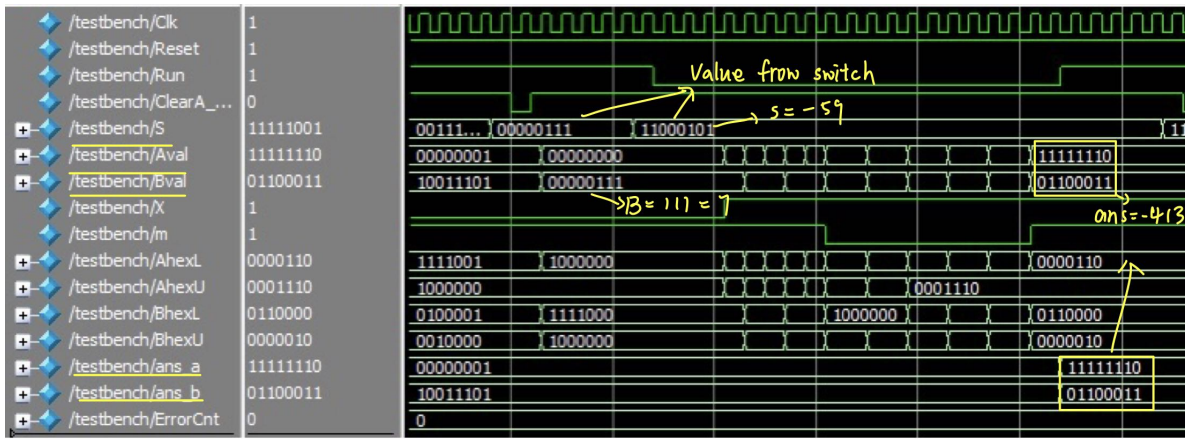

Figure 11: Waveform of (-59)*7

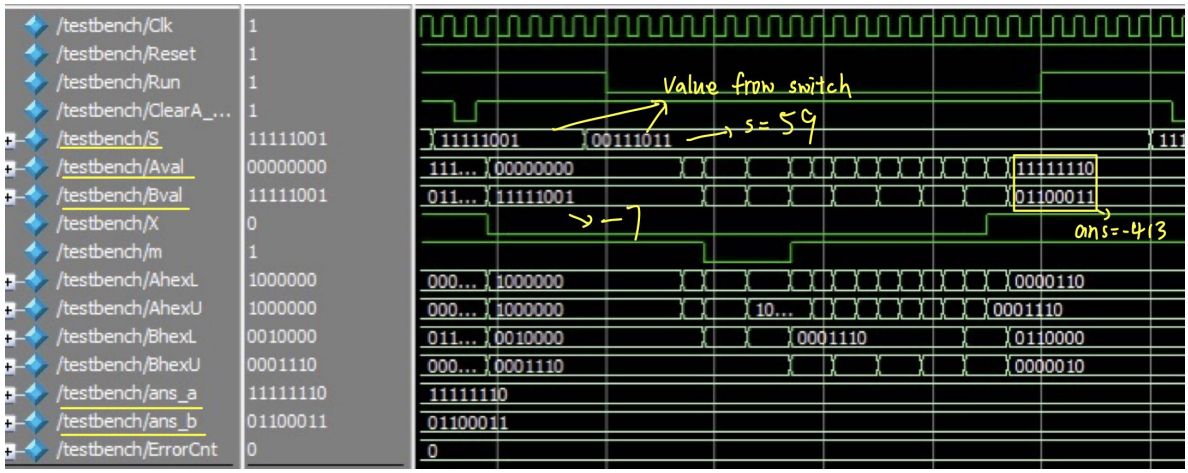

Figure 12: Waveform of 59*(-7)

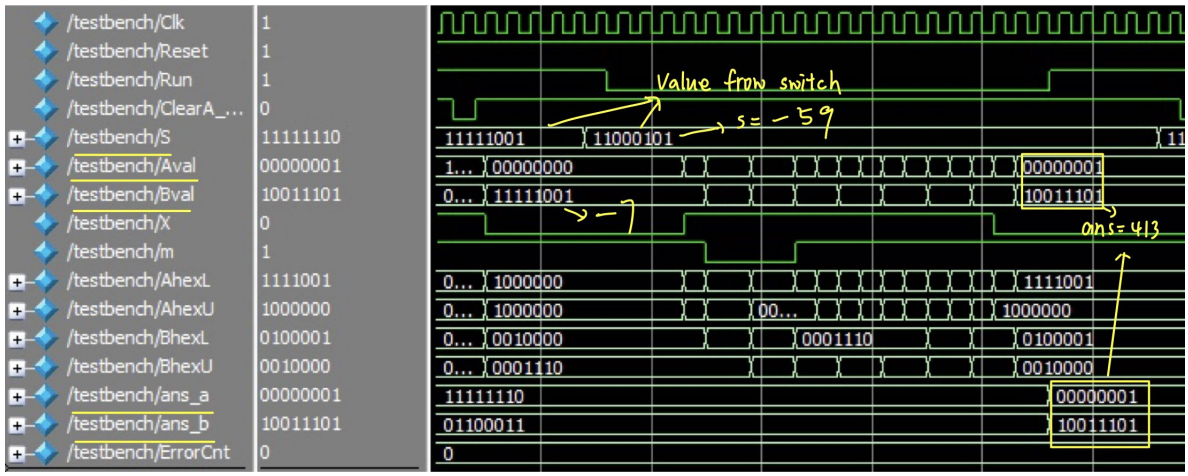

Figure 13: Waveform of (-59)*(-7)

4 Answers to two post-lab questions1

4.1 Fill in the table shown in 5.6 with your design's statistics

LUT	100
DSP	0
Memory(BRAM)	0
Flip-Flop	64
Frequency	71.95 MHz
Static Power	98.51 mW
Dynamic Power	4.17 mW
Total Power	147.08 mW

Table 1: Design Resources and Statistics

4.2 Write down several ideas on how the maximum frequency of your design could be increased or the gate count could be decreased

- To increase the maximum frequency, we can replace our carry-ripply adder to more efficient carry-select adder or carry-lookahead adder, because their design reduce delays and run some computation at the same time.

- To decrease our gate counts, we can put 8-bit register(A) and one flip-flop(X) together to get a 9 bit register. Therefore, the additional gates for flip-flop can be emitted and gates will get decreased.

4.3 Answer the following questions

4.3.1 What is the purpose of the X register. When does the X register get set/cleared?

X register is used to store the extension of sign bit of the result so that the sign can remain correct when doing shift operation. X register gets set/cleared when the most significant number of result alternates from 0 and 1 after doing add operation.

4.3.2 What are the limitations of continuous multiplication? Under what circumstances will the implemented algorithm fail

When the result of continuous multiplication result needs 8 more bits, the algorithm will fail because every time doing new multiplication, A register will be cleared, if the last result has more than 8 bits, the bits in A will be changed to zero.

4.3.3 What are the advantages and disadvantages of the implemented multiplication algorithm over the pencil-and-paper method discussed in the introduction?

Advantages are that we only need 9 bit and 8 bit register to do the multiplication, we do not need to add one multiplicand one at a time and add total the number of another multiplicand times, therefore it saves a lot of time. The disadvantages are that the circuit is very complex compared with simple add algorithm, therefore we need more gates and more registers which increase hardware cost. Besides, we cannot get more than 8 bit results during the continuous multiplying.

5 Conclusion

5.1 Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it

We design a multiplier for two 8-bit 2's compliment numbers and then run it on the DE2 FPGA board. Using this multiplier, we accomplish a single multiplication or continuous multiplication. After loading B with switch and set switch for A, and push Run button, the $result = A \times B$ will show on the DE2 board. Every parts in our design work well. But we meet many bugs when we try to accomplish our design and work it out. For example, because we need to clear A register every time before we run continuous multiply so we add clear A instruct at our start state. when we simulate, we find that if we didn't keep on pressing the button, the data in register A will be cleared. To fix it, we add another display state after Halt state to display our result and if we press run button, we will do continuous multiply after that.

5.2 Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.

The table which shows how bits flow in the register is really helpful for our understanding. And the block diagram is also clear. However, I still have some suggestions for the written material. I think the description of ClearA_loadB signal is not so clear and we pay a lot of time to decide when we need only clearA and when we need ClearA_loadB, and how to use them. Besides, the pencil-and-paper method's bit is not aligned, which confused me a lot at first.